EXPLOITING UNLABELED DATA IN CONCEPT DRIFT LEARNING

Dwi Hendratmo Widyantoro
School of Electrical Engineering and Informatics
Bandung Institute of Technology
Email: dwi@if.itb.ac.id

ABSTRACT: Learning unlabeled data in a drifting environment still receives little attention. This paper presents a concept tracker algorithm for learning concept drift that exploits unlabeled data. In the absence of complete labeled data, instance classes are identified using a concept hierarchy that is incrementally constructed from data stream (mostly unlabeled data) in unsupervised mode. The persistence assumption in temporal reasoning is then applied to infer target concepts. Empirical evaluation that has been conducted on information-filtering domains demonstrates the effectiveness of this approach.

Keywords: concept drift learning, unlabeled data, persistence assumption.

INTRODUCTION

Combining labeled and unlabeled data in concept learning has gained its popularity since the past few years. With the proliferation of information on the Internet, unlabeled data become more widely available but acquiring labeled data still poses a bottleneck [9]. As such, unlabeled data can provide information that fills the gap in the labeled data.

There are several existing algorithms for learning unlabeled data. Expectation Maximization (EM) algorithm [4], for example, works by iteratively revising a hypothesis that maximizes its likelihood. Co-training is a more recent method that exploits sets of distinct features in unlabeled data set [2]. Blum and Chawla develop an algorithm based on finding the minimum cuts in graph for combining labeled and unlabeled data [3]. Although the effectiveness of these methods is not questionable, these methods mainly assume the stability of learned concepts. Therefore, this assumption is not suitable for inducing target concepts that change over time such as in concept drift learning.

A number of practical algorithms for learning concept drift [10][13][14][15][16] as well as its theoretical characterizations [1][7] have been developed in the past. The existing concept drift learning algorithm, however, do not incorporate unlabeled data and assume the availability of numerous labeled data in order to achieve performance at satisfactory levels. The assumption is certainly not realistic in real world application. This paper proposes a concept tracker algorithm that utilizes unlabeled data for learning concept drift in the presence of incomplete labeled data.

PRELIMINARY OBSERVATIONS

This section presents empirical and theoretical observations that motivate the approach described in this paper. Prior to discussing preliminary experiment results, it will briefly describe the experiment setup for obtaining the empirical finding.

Following the standard in concept drift learning, the goal of experiments was to observe the performance of system as target concepts change over time. Accordingly, the system was presented with a stream of data to learn sequentially, and its performances were measured on a fixed test set with respect to current target concepts at regular intervals after processing *m* consecutive instances. A period of incremental learning on the *m*-instance sequence and system's performance measurement was then called a *tracking cycle*. The target concepts were made stable for twenty tracking cycles before changed to a different target concept set. Section experiments provides further experiment details.

Figure 1 depicts the system's performances over time from learning with completely and partially labeled data stream. The FULL-LABELED-DATA in the figure denotes a system that receives a new sequence of four labeled instances at each tracking cycle. Labeled data sets are given to the PARTIAL-LABELED-DATA system only when target concepts are about to change (i.e. at tracking cycles 1, 21 and 41), amounting to five percent as many labeled data as learned by the FULL-LABELED-DATA system.

The performances over time given by the FULL-LABELED-DATA system, as shown by Figure 1, are typical in concept drift learning. The system's accuracy drops when a concept drift occurs and then recovers over the next tracking cycles as the system receives more labeled data for learning new target concepts. In contrast, the PARTIAL-LABELED-DATA system has no way of improving its performances in the absence of labeled data. It also suffers from being unable to accurately learn the new target concepts from learning with partially labeled data. Due to

applying the same test set, its performances are constant at low accuracies and tracking cycles during which the target concepts are stable.

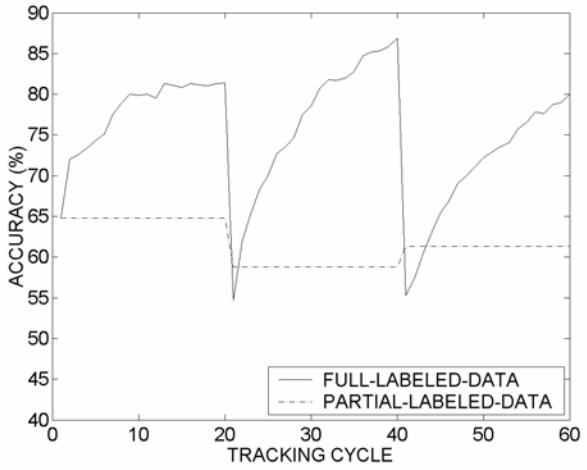

Figure 1. System's performance over time on fullly and partially labeled data sets

The above empirical observations are also well justified by existing theoretical finding. In concept drift learning, the *drift rate* Δ is an essential parameter and denotes the probability that two successive target concepts disagree on a randomly drawn example [7], e.g. **Prob**($f(x_m) \neq f(x_{m+1})$). Hence, a slower drift rate corresponds to learning from a data stream whose target concepts change less frequently, and vice versa. Helmbold and Long provide theoretical bounds on the allowable drift rates that guarantee tractability with an error of at most ε as follows:

$$\Delta \leq \frac{c\varepsilon^2}{d \ln(1/\varepsilon)} \qquad (1)$$

where $c > 0$, and d is the *Vapnik-Chervonenkis* dimension of a concept/hypothesis. Because c and d values are constants, the bounds imply that the tracking problem is more difficult (i.e. producing higher error rates) on learning with fewer labeled instances per target concept (i.e. higher drift rates).

A NOVEL APPROACH

Reducing the rate of drift according to Equation (1) is apparently the only option for improving the performance of a concept drift learner. One of possible strategies for reducing the drift rate as considered in this paper is to fill the gap in labeled data with relevant unlabeled data. The following describes an algorithm that infers the current label of each labeled instance, based on the instance classes, and expands it with relevant unlabeled data.

Concept Tracker Algorithm

Let's now assume that the associations between an instance and its concept class somehow can be identified. Let X be the instance spaces and C be the concept class spaces. The following defines two functions needed to access the instance-class associations:

- δ: $X \rightarrow C$ is a function that, given an instance, returns the instance class.
- ε: $C \rightarrow \mathrm{P}(X)$ is a concept instantiation function that returns a set of instances covered by a concept class.

Let $Q = \{+,-\}$ be a set of labels and let $IS = \langle (x_1,q_1), \ldots, (x_n,q_n) \rangle$ for each $x_i \in X$ and $q_i \in Q$ denote a sequence of n labeled instances (e.g., data stream) where an instance on the left side arrives earlier. In Information Filtering domain, for example, an instance x_i is a text document (or a Web page), and a label (or feedback) is given to the document whether it is relevant ("+") or irrelevant ("–"). Figure 2 describes a Concept Tracker algorithm that takes *IS* as input and outputs a set of labeled concept classes RC and a set of expanded labeled instances RI.

Input: $IS=\langle (x_1,q_1), \ldots, (x_n,q_n) \rangle$ a sequence of labeled instances where q_i is the label of instance x_i.
Output: **RC**, a set of labeled concept classes.
RI, a set of expanded labeled instances.
Step 1. Generate a stream of labeled concept classes by replacing each instance in *IS* with its corresponding concept.
Let $CS = \langle (c_1,q_1), \ldots, (c_n,q_n) \rangle$ such that $c_i = \delta(x_i)$ for each $(x_i,q_i) \in IS$ and $c_i \in C$.
Step 2. Infer the label of each distinct concept class.
Let $\mathbf{RC} = \{(c_i, q_i)\}_{c_i \in \mathbf{C}}$ where **C** is a set of distinct concepts occurring in the stream *CS* and q_i is the label assigned to the most recent c_i in *CS*.
Step 3. Instantiate each distinct concept class in **RC** by replacing the concept class with all instances it covers.
Let $RI = \{(\{x_j\}_{X_j \in \varepsilon(C_i)}, q_i)\}_{(c_i,q_i) \in RC}$

Figure 2. Concept Tracker Algorithm (see the description of algorithm in text).

The foundation for reasoning the label of each concept class in Step 2 is that of *persistence assumption*, which states that once a fact becomes true it remains true thenceforth until the fact is negated [5]. The persistence assumption enables reasoning the label of each concept class at any given time. Since the concept drift learning typically refers to current

target concepts, the class labels can be simply inferred from the most recent labels assigned to concept classes. The following presents the illustrating example for the concept tracker algorithm.

Suppose $IS = \langle(x_1,+), (x_2,+), (x_3,–), (x_4,–), (x_5,–), (x_6,–), (x_7,+), (x_8,+)\rangle$ (the input of the concept tracker algorithm) is the original data stream (e.g., a set of relevance feedback documents). Let $\{x_1, x_3, x_8, x_{12}, x_{13}\}$ be a set of instances belong to class c_1 such that $\delta(x_1)=\delta(x_3)=\delta(x_8)=\delta(x_{12})=\delta(x_{13})=c_1$ and $\varepsilon(c_1)=\{x_1, x_3, x_8, x_{12}, x_{13}\}$. Similarly, let $\{x_2, x_6, x_9, x_{11}\}$ and $\{x_4, x_5, x_7, x_{10}, x_{14}\}$ be the sets of classes belong to c_2 and c_3 respectively. Classes c_1, c_2, c_3 can represent document topic categories in Information Filtering domain. The stream of labeled classes generated by Step 1 in Figure 2 is thus $CS = \langle(c_1,+), (c_2,+), (c_1, –), (c_3,–), (c_3,–), (c_2, –), (c_3,+), (c_1,+)\rangle$. Step 2 will generate distinct concepts **RC**$=\{(c_1,+), (c_2, –), (c_3,+)\}$ based on the most recent labels assigned to concept classes (i.e., "+" is the most recent label assigned to classes c_1 and c_3). Finally, all distinct classes in Step 3 are instantiated to produce more complete sets of labeled instances **RI**$=\{(\{x_1, x_3, x_8, x_{12}, x_{13}\},+), (\{x_2, x_6, x_9, x_{11}\}, –), (\{x_4, x_5, x_7, x_{10}, x_{14}\},+)\}$.

Any concept learner suitable for a domain can then be applied to learn the instance set **RI**. The learner can also take advantage the identified concept classes **RC** if necessary. When the input *IS* changes, the outputs must be entirely re-generated and re-learned. Despite the simplicity of the Concept Tracker algorithm, its effectiveness depends largely on the reliability of the δ function for identifying the instance classes. The next section provides the detail of the approach in handling unlabeled data in order to realize δ and ε functions.

Identification of Instance Classes

This paper uses a concept hierarchy for establishing the association between an instance and its concept class. The concept hierarchy is incrementally built from labeled and unlabeled data observed from the stream in unsupervised mode. Thus, during the course of learning it grows dynamically as it sees more instances. The cluster hierarchy is basically a tree structure with the following characteristics: (1) all leaf nodes represent instances and (2) all internal nodes represent concepts that generalize those of their descendants. Given a concept c, which is an internal node in the concept hierarchy, ε(c) returns a set of leaf nodes that are descendant of node *c*. Distinct concepts emerge from concepts in the hierarchy that distinctively partition instances. Obviously, these distinct concepts are the most appropriate concepts for representing the classes of all instances underneath.

A recently developed concept formation system is employed for constructing the concept hierarchy needed [17][18]. One of the concept properties generated by the system is *concept density*, which is calculated from the average distance to the nearest neighbor among the child concepts. The concept density in the hierarchy tends to decrease along the path to the root. At any time, a set of distinct, non-overlapping concepts can be identified from the concept hierarchy by thresholding the concept density information. The distinct concepts identified, however, could change as the concept hierarchy is updated over time. The process for identifying distinct concepts is similar to that of in *proximity dendogram* cutting [8] that identifies clusters according to dissimilarity levels.

The density threshold for identification of concept class is empirically determined using a validation set. First, a concept hierarchy is incrementally built from a stream of data in the validation set. Because the class of each instance is known in the validation set, distinct concepts that correspond to classes in the validation set can be accurately identified from the concept hierarchy. The threshold is then calculated from the densities of these distinct concepts. More specifically, let H be the concept hierarchy generated from the validation set containing a set of concept classes T. Let $c_t \in H$ be a concept that corresponds to concept class $t \in T$. Furthermore, let $\varepsilon(c)$ be a set of instances covered by concept c. The concept c_t is identified from H by:

$$c_t = \arg\max_{c \in \mathcal{H}} \left\{ \sum_{x \in \varepsilon(c)} m_{x,t} - \sum_{u \in T-\{t\}} \sum_{x \in \varepsilon(c)} m_{x,u} \right\} \quad (2)$$

where $m_{x,t}$ is a binary matching function such that $m_{x,t}$ =1 if the class of instance x is t, or 0 otherwise. Hence, c_t maximizes the difference between the numbers of instances that are on t class and non-t classes.

For example, let $T=\{t_1, t_2\}$ be the set of classes in the validation set where $\varepsilon(t_1) = \{x_1, x_2, x_3\}$ and $\varepsilon(t_2) = \{x_4, x_5, x_6\}$. Let $\{c_1, c_2, c_2\} \subset H$ be a set of non-root concepts, which is a subset of a concept hierarchy H generated from the same instances in the validation set. Suppose $\varepsilon(c_1) = \{x_1, x_2\}$, $\varepsilon(c_2) = \{x_3, x_4\}$ and $\varepsilon(c_3) = \{x_5, x_6\}$. To find the corresponding concept class t_1, one can use Equation (2) to calculate the difference between the numbers of instances that are on t_1 and on t_2 for instances of c_1, c_2 and c_3, and then select the one that maximizes this difference. These differences are $(2 – 1)=1$ for c_1, $(1 – 1)=0$ for c_2, and $(0 – 2)= –2$ for c_3. Hence, c_1 is the corresponding concept for class t_1 because c_1 has the maximum difference. Applying the same procedure, one will find that c_3 is the corresponding concept for class t_2.

Now, let $\mu(c_t)$ be the average distance to the nearest neighbor among c_t's child nodes; μ represents the concept density in the concept formation system employed. A higher μ value corresponds to a lower-density concept, and vice versa. Thus, the μ values tend to increase along the paths from leaf nodes to the root. Several alternatives can be used to determine the threshold value, among of these are θ_{min} and θ_{max} as defined below:

$$\theta_{\min} = \frac{1}{|T|}\sum_{t\in T}\mu(c_t) \quad (3)$$

$$\theta_{\max} = \frac{1}{|T|}\sum_{t\in T}\max\left\{\mu(c_t), \frac{\mu(c_t)+\mu(c_t\text{'s parent})}{2}\right\} \quad (4)$$

The threshold value of θ_{min} is calculated by averaging the density values of corresponding concepts found in the concept hierarchy. The threshold value of θ_{max} is averaged over either the density value of the corresponding concept or the average density among the corresponding concept and its parent, whoever gives the maximum value. The threshold θ_{min} poses the risk of overfitting the concept class while θ_{max} could underestimate the concept density. To avoid both of these potential problems, this paper applies thresholding stochastically for any density value between θ_{min} and θ_{max}.

Let $x=c_0$ be an instance (e.g. a leaf node in the concept hierarchy) and $A_X = \{c_0, c_1, \ldots, c_n\}$ such that c_j is c_{j-1}'s parent and c_n is the root. Define $\varphi: X \times C \rightarrow \{1, 0\}$ to be a binary function such that:

$$\varphi(x, c_j) = \begin{cases} 1 & \textbf{iff } \psi(c_i)=1 \textbf{ for} \forall c_i \in A_x \textbf{ and } j \geq i \geq 0 \\ 0 & \textbf{otherwise} \end{cases} \quad (5)$$

where

$$\psi(c_i) = \begin{cases} \mathbf{1} & \textbf{for } \boldsymbol{\mu}(c_i) \leq \theta_{\min} \\ 1 & \textbf{with Prob}(\psi(c_i)=1) = \dfrac{\theta_{\max} - \boldsymbol{\mu}(c_i)}{\theta_{\max} - \theta_{\min}} \\ & \qquad \textbf{for } \theta_{\min} < \boldsymbol{\mu}(c_i) < \theta_{\max} \\ 0 & \textbf{for } \boldsymbol{\mu}(c_i) \geq \theta_{\max} \end{cases}$$

Thus, the function $\varphi(x, c_j)$ returns 1 if the densities of all concepts in the path from the leaf node x to its ancestor c_j are within the threshold (i.e., less than or equal to θ_{min}). If the concept density is between θ_{min} and θ_{max}, then it will be considered as within the threshold with a probability as defined in Equation (5). A concept c is said to be *compact* if only if $\varphi(x, c) = 1$ for every $x \in \delta(c)$. A *distinct concept* representing the classes of all instances that are its descendants in the concept hierarchy can be identified from a *compact concept* whose parent is not *compact*. Similarly, the concept class of an instance x or $\delta(x)$ can be identified from its ancestor that is a distinct concept.

EXPERIMENTS

This section describes a set of experiments that evaluate the concept tracker algorithm on information-filtering domains. The performances of existing concept learners that learn a sequence of partial labeled instances will be used as the baseline. The effectiveness of the concept tracker algorithm can then be observed from the performance gains achieved, over the baseline performances, from applying the same learner but using the data set produced by the proposed algorithm. In addition, the experiments also investigate the effects of various biases employed by the concept learners.

Table 1. Summary of topics and documents for evaluation.

Topics (Concept Classes)	Training Set	Validation Set	Test Set
Trade, Coffee, Crude, Sugar, Acq	400*	100**	100**
Others (54 topics)	–	–	427

t *80 and **20 documents per topic category.

Data Set

A subset of the Reuters-21578 1.0 collection was used in all experiments. The documents were pre-processed by removing stop words, stemming the remaining words and extracting them as individual terms. The document terms were weighed using the TF-IDF method [12]. Table 1 summarizes the topics and documents selected for the evaluation. The document topics represent concept classes. The validation set, as described in Section Identification of Instance classes was used to empirically determine the density threshold.

Given: a data stream *S* generated from the training set.
Initialization:
  IS = ⟨∅⟩, i.e. the sequence of labeled instances.
  H = ∅, the concept hierarchy.
  Determine the density threshold from the validation set.
For each *tracking cycle i* = {1 … *K*}
1. Process incrementally the i^{th} *m*-instance sequence from *S*.
  For each instance *x* from the *m*-instance sequence
    Update *H* incrementally to incorporate *x*.
    If the label *q* of *x* is available,
      Concatenate ⟨(*x*, *q*)⟩ to the end of *IS*.
2. Identify distinct concepts in H.
3. Execute the *Concept Tracker* algorithm to generate **RC** and **RI** sets from current values of *IS* and *H*.
4. Run a selected classifier (concept learner) to learn the stream generated from instances in **RI** and measure the accuracy of the learned concepts on the test set.

Figure 3. Experiment procedure for evaluating concept tracker algorithm

Procedure and Tracking Tasks

Figure 3 summarizes the experiment procedure, which is slightly modified from the one described in Section Preliminary Observations. The procedure accommodates both the incremental and batch processes needed in the proposed framework. The same test set was used to measure the system's performance on each tracking cycle and is calculated as follows. All instances in the test data were ranked based on their similarities to learned concepts. The system's accuracy was then measured by calculating the percentage of target test instances ranked within the top *n* data. Specifically, let *P* be the number of target instances that appear in the top *n* data ranked by a system. The accuracy of the system was calculated by the equation below:

$$\text{Accuracy} = \frac{P}{\sum_i T_i} \times 100\% \qquad (6)$$

where T_i were the numbers of target instances being considered in the current tracking cycle and $n = \sum_i T_i$ was the total number of target instances in the test data.

The data streams were generated according to a tracking task, a scenario that described the evolution of target concepts over time. Tables 2 and 3 provide two tracking tasks used in the experiments. Each column in the tables describes the number and the class of instances in the *m*-instance sequence that is processed at each tracking cycle. In Table 2, for example, tracking cycles 21–40 processed two-instance sequences; each contains one TRADE instance and one COFFEE instance ordered randomly in the sequence. Each tracking cycle used a new set of instances from the training set that had not been seen. Information regarding the instance classes was not told to the system.

Table 2. The first tracking task: learning the evolution of a single target concept. Note that TR=TRADE, CO=COFFEE, CR=CRUDE, SU=SUGAR and AC = ACQ as described in Table 1.

Tracking Cycle				
1 – 20	21 – 40	41 – 60	61 – 80	81 – 100
(TR, +) (*m*=1)	(TR, –) (CO, +) (*m*=2)	(CO, –) (CR, +) (*m*=2)	(CR, –) (SU, +) (*m*=2)	(SU, –) (AC, +) (*m*=2)

Table 3. The second tracking task: learning the evolution of multiple target concepts.

Tracking Cycle		
1 – 20	21 – 40	41 – 60
(Trade, +) (Coffee, +) (CRUDE, +)	(Trade, –) (Coffee, +) (CRUDE, +) (SUGAR, +)	(Coffee, –) (Crude, +) (SUGAR, +) (ACQ, +)
(*m*=3)	(*m*=4)	(*m*=4)

Table 4. The sequence of partial labeled instances.

Amount of Labeled Data	Tracking Cycles
5 Percent	1, 21, 41, …
10 Percent	1, 11, 21, 31, 41, …
25 Percent	1, 6, 11, 16, 21, 26, 31, 36, 41, …

For simplicity, target concepts were made stable for periods of twenty tracking cycles. Labeled instances that marked the beginning of change in target concepts were given at the first tracking cycles during the twenty-tracking cycle periods, i.e. at tracking cycles 1, 21, 41 and so on. Concept classes with positive (+) labels indicated the desired target concepts at the respective tracking cycles. Positive labeled instances were used to establish new (or emphasize the existing) target concepts. The negative labels were used to negate previously established target concepts. For example, a positive TRADE instance in Table 2 was given during the first tracking cycle to establish the TRADE target concept. The labeled instances provided during the 21st tracking cycle contained one positive COFFEE instance and one negative TRADE instance, which changed the target concept from TRADE to COFFEE.

The experiments were varied using data streams containing 5, 10 and 25 percents of labeled instance sets. Table 4 describes the tracking cycles at which the labeled instance sets were provided in each stream. Since the results depended on the order of instances in the stream, the accuracy at each tracking cycle was averaged over ten runs. Each run used a different stream of instance sets but it represented the same tracking task.

Varying the Concept Learner

The experiments were also conducted by varying the concept learner (or classifier) that learned the data set **RI** generated by the concept tracker algorithm. In particular, we considered three classifiers from information retrieval techniques. The first classifier was the incremental version of the Rocchio's relevance feedback algorithm [11]. The classifier learned concept classes in a single-descriptor representation. A descriptor *d* is a list of feature and its weight pairs. For each labeled instance *d* in the stream, it incrementally updated its descriptor as follows:

$$d_{new\ Rocchio} = d_{old\ Rocchio} + \alpha \cdot d \qquad (7)$$

where $\alpha = 0.1$ for a positive labeled instance and $\alpha = -0.9$ for a negative labeled instance. In previous work [16] these settings were optimal for learning changing concepts in the domain.

The second classifier was multiple ROCCHIO algorithm, or M-ROCCHIO for short. It maintained

multiple descriptors; each was updated using the same incremental ROCCHIO algorithm as above. The M-ROCCHIO classifier also utilized the concept classes **RC** identified by the concept tracker algorithm. Accordingly, each descriptor was used to learn only instances of the same concept class.

The third classifier was MTDR (**M**ultiple **T**hree-**D**escriptor **R**epresentation) algorithm [16]. This classifier was similar to M-ROCCHIO in that multiple concept classes were represented in multiple descriptors. Unlike M-ROCCHIO, the MTDR classifier used its own built-in algorithm for identifying the concept classes.

The ROCCHIO and MTDR classifiers learned a single stream. The M-ROCCHIO algorithm learned multiple streams; each stream was learned by a separate descriptor and contained instances of the same class with respect to concept classes **RC**. All streams were generated from the instance set **RI** and were ordered based on the instance arrival times.

The three classifiers performed a ranking based classification on a given test set. The score of a new instance was calculated according to the cosine similarity metric [12] with respect to the learned concepts. For those with multiple descriptor representations (e.g. M-ROCHHIO and MTDR), the score was based on the highest similarity akin to the nearest neighbor classification.

Results

Figures 4 and 5 show the performance over time from learning data streams that contain five percent of labeled instances on the second tracking task. The PLD+UNLABELED-DATA results were obtained by running the classifiers using the instance sets generated by the concept tracker algorithm. In the absence of labeled data, as shown by the figures, the performances improve as expected as more relevant unlabeled data become available from the streams.

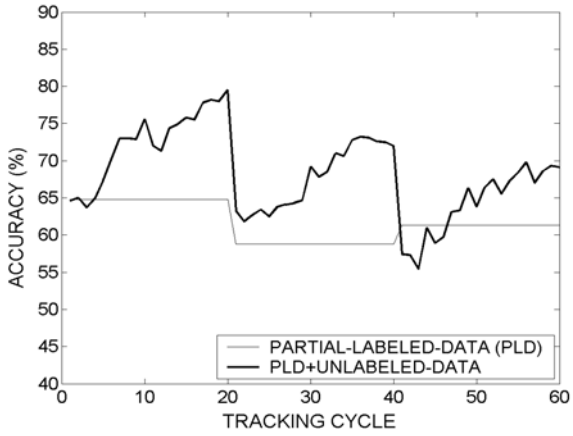

Figure 4. Rocchio's accuracies on the 2nd tracking task.

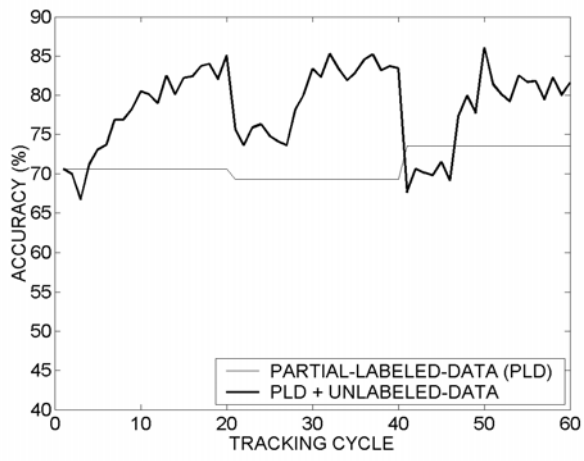

Figure 5. MTDR's accuracies on the 2nd tracking task.

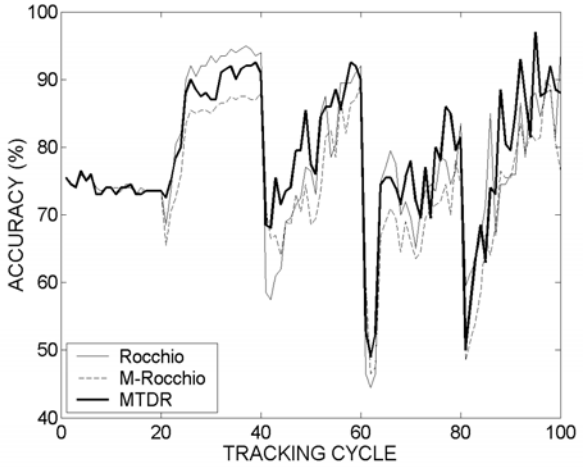

Figure 6. Performance comparison on the 1st tracking task.

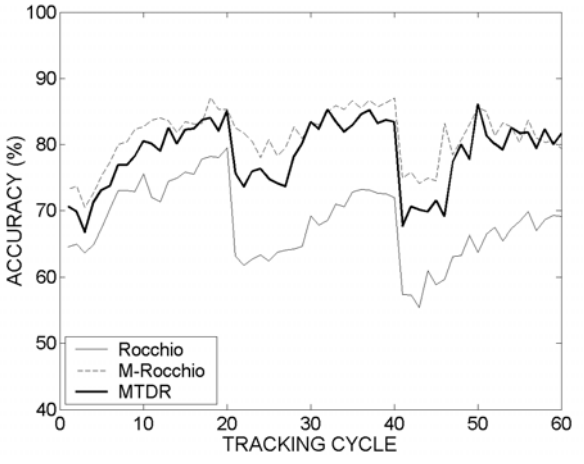

Figure 7. Performance comparison on the 2nd tracking task.

Table 5. The summary of experiment results on the first tracking task. The numbers following the "+"s within parentheses denote the performance gains achieved by incorporating unlabeled data over the baseline performances on the same amount of labeled data observed and the same classifier employed.

Amount of Labeled Data	Average Accuracy (%)				
	MTDR		Rocchio		M-Rocchio
	Labeled Data (Baseline)	Labeled + Unlabeled Data	Labeled Data (Baseline)	Labeled + Unlabeled Data	Labeled + Unlabeled Data
5 Percent	63.00	78.91 (+25.3%)	46.90	77.66 (+65.6%)	74.59
10 Percent	74.10	81.01 (+9.3%)	60.75	82.13 (+35.2%)	79.24
25 Percent	79.07	82.14 (+3.9%)	70.52	82.00 (+16.3%)	78.54
100 Percent	86.96	–	79.82	–	–

Table 6. The summary of experiment results on the second tracking task.

Amount of Labeled Data	Average Accuracy (%)				
	MTDR		Rocchio		M-Rocchio
	Labeled Data (Baseline)	Labeled + Unlabeled Data	Labeled Data (Baseline)	Labeled + Unlabeled Data	Labeled + Unlabeled Data
5 Percent	71.13	78.50 (+10.4%)	61.63	68.11 (+10.5%)	81.26
10 Percent	75.61	80.93 (+7.0%)	68.06	69.85 (+2.6%)	83.44
25 Percent	77.72	80.87 (+4.1%)	69.07	71.56 (+3.6%)	81.89
100 Percent	86.07	–	75.14	–	–

Using the same data sets provided by the concept tracker algorithm, the system's performances could be significantly affected by the classifier employed. Figure 6 depicts the performance comparison among the three classifiers on the first tracking task, i.e. tracking the evolution of single target concept. All classifiers have the capability for learning at least a single target concept so that their performances are expected to be comparable to one another on this task. This expectation, as indicated by Figure 6, is satisfied in the case of the ROCCHIO and MTDR performances. The M-ROCCHIO performances are slightly worse than the others.

Figure 7 shows the system's performances on the second tracking task, i.e. tracking the evolution of three target concepts. In contrast to the first tracking task, the M-ROCCHIO classifier performs better than the other two classifiers. The observation that the ROCCHIO performances are relatively much worse is not surprising because this classifier lacks the representational power needed for learning multiple target concepts.

Table 5 and Table 6 summarize the experiment results obtained by averaging the system accuracies from the beginning of tracking cycle to the end. The tables reveal that incorporating unlabeled data always improves the average accuracies with respect to the baseline performances. Increasing the portion of labeled data in the stream does not seem to result in a full recovery of performance as achieved from learning with complete (100 PERCENT) labeled data. However, the fact that highest performance gains are obtained from learning with the least amount of labeled data is very encouraging because this is the case that is being addressed by the paper.

Further observations reveal that the data sets generated by the concept tracker algorithm contain 19% noise in the first tracking task and 5% in the second tracking task on the averages. These explain why M-ROCCHIO performs worst in the first task but it performs best in the second task. The M-ROCCHIO algorithm utilizes the concept classes identified by the concept tracker algorithm verbatim so that the quality of data set generated by the concept tracker algorithm directly affects the M-ROCCHIO performance. On the other hand, the MTDR classifier independently learns the class of each instance that makes this classifier less susceptible to noise.

RELATED WORKS

STAGGER was the first incremental learning system that dealt with concept drift problem by thresholding concept weights [13]. Widmer and Kubat developed a family of FLORA learning algorithms [14]. The system employed an adaptive window adjustment heuristic for flexibly responding to concept drift, reused previously learned stable concepts and

handled the presence of noise. In more recent work, Widmer exploited contextual attributes on a fixed window size for tracking context changes through meta-learning [15]. SPLICE was perhaps the first concept drift system that employed an off-line approach, e.g. using Quinlan's C4.5, for identifying hidden contexts [6]. Tracking the evolution of user interests is an instance problem of concept drift learning in information-filtering domains. Klinkenberg and Joachims also employed an adaptive window adjustment heuristic for learning changing user interests but used support vector machine (SVM) as the underlying classifier [10]. Their method worked on an input stream that arrived in batches. Learning the dynamics of user preference had also been developed using three-descriptor representations [16]. The algorithm employed an implicit windowing technique (i.e. by decaying older examples), which combined large and small windows.

CONCLUSIONS

This paper has described the computational framework for incorporating unlabeled data in concept drift learning and has presented its evaluation on information-filtering domains. In the absence of labeled data, the system improves its performance over time as relevant unlabeled data become available. The framework offers a flexible architecture that allows the classifier to be tailored to target applications. Although the soundness of the concept tracker algorithm is obvious, its effectiveness depends on the method for identifying instance classes, the concept formation system used for building the concept hierarchy, and the classifier employed for concept learning. This leaves a lot of room for improvement.

Acknowledgement

The author would like to thank the reviewers for their useful comments and suggestions.

REFERENCES

1. Bartlett, P.L., David, S.B., & Kulkarni, S.R., Learning Changing Concepts by Exploiting the Structure of Change. *Machine Learning*, 41, 2000, pp.153-174.
2. Blum, A., & Mitchell, T. M., Combining Labeled and Unlabeled Data with Co-Training. *Proceedings of the 11th Annual Conference on Computational Learning Theory*, 1998, pp. 92-100.
3. Blum, A., & Chawla, S., Learning from Labeled and Unlabeled Data using Graph Mincuts. *Proceedings of the 18th International Conference on Machine Learning*, 2001, pp. 19-26.
4. Dempster, A. P., Laird, N. M., & Rubin, D. B., Maximum Likelihood from Incomplete Data via the EM Algorithms. *Journal of the Royal Statistical Society*, Series B., 39(1), 1977, pp. 1-38.
5. Gabbay, D. M., Hogger, C. J., & Robinson, J. A. *Handbook of Logic in AI and Logic Programming: V4. Epistemic and Temporal Reasoning*. New York: Oxford University Press, 1995.
6. Harries, M.B., Sammut, C., & Horn, K., Extracting Hidden Context. *Machine Learning*, 32(2), 1998, 101-128.
7. Helmbold, D. P. & Long, P. M., Tracking Drifting Concepts by Minimizing Disagreement. *Machine Learning*, 14, 1994, pp. 27-45.
8. Jain, A.K. & Dubes, R.C., *Algorithms for Clustering Data*. Englewood Cliffs: Prentice Hall, 1988.
9. Jansen, B. J., Spink, A., & Saracevic, T., Real life, real users and real needs: A study and analysis of users queries on the Web. *Information Processing and Management,* 36 (2), 2000, pp. 207-227.
10. Klinkenberg, R., & Joachims, T., Detecting Concept Drift with Support Vector Machine. *Proceedings of the 17th International Conference on Machine Learning*, 2000, pp. 487-494.
11. Rocchio, J.J., Relevance Feedback in Information Retrieval. In G. Salton, *The SMART Retrieval System: Experiments in Automatic Document Processing* Englewood Cliffs: Prentice-Hall, 1971. pp. 313-323.
12. Salton, G., & McGill, M. J., *Introduction to Modern Information Retrieval*. N.Y.: McGraw-Hill, 1983.
13. Schlimmer, J.C., & Granger, R.H., Beyond Incremental Processing: Tracking Concept Drift. *Proceedings of the 5th National Conference on Artificial Intelligence*, 1986, pp. 502-507.
14. Widmer, G., & Kubat, M., Learning in the Presence of Concept Drift and Hidden Contexts. *Machine Learning*, 23(1), 1996, pp. 69-101.

15. Widmer, G., Tracking Context Changes through Meta-Learning". *Machine Learning*, 27(3), 1997, pp. 259-286.

16. Widyantoro, D. H., Ioerger, T. R., & Yen, J., Learning User Interest Dynamics with a Three-Descriptor Representation. *Journal of the American Society for Information Science*, 52(3), 2001, pp. 212-225.

17. Widyantoro, D. H., Ioerger, T. R., & Yen, J., An Incremental Approach to Building a Cluster Hierarchy. *Proceedings of the 2nd IEEE International Conference on Data Mining*, 2002.

18. Widyantoro, D.H., Exploiting the Homogeneity of Density in Incremental Hierarchical Clustering. *Proceedings ITB Series B: Engineering Scienc*e, Vol 38B, No. 2, 2006, pp. 79-99.